CS6.401 Software Engineering (Spring 2024)
Project: 1, Team no: 27

Team Members:

Ronak Redkar	2023201046
Vivek Ram	2023201055
Naitik Kariwal	2023201044
Karun Choudhary	2023201038
Jayank Mahaur	2023201043

UML Description 1A: Book Addition & Display Subsystem

User Class

- Represents a user registered in the system.
- Manages user properties such as ID, username, password, email, etc.
- Provides methods to get and set these properties.

BookResource Class

- Acts as an interface for users to interact with the book-related functionalities in the system.
- Allows users to add, delete, update, retrieve, and manage books.
- Provides methods for various book operations like adding a book, updating its details, listing books, etc.

BookDao Class

- Manages the persistence layer for books in the system.
- Responsible for creating, retrieving, and updating book entities.
- Provides methods to interact with the database or data storage to perform CRUD operations on books.

UserBookDao Class

- Handles the mapping between users and the books they own.
- Stores information about which user owns which books.
- Provides methods to create, retrieve, and delete user-book mappings.

UserBook Class

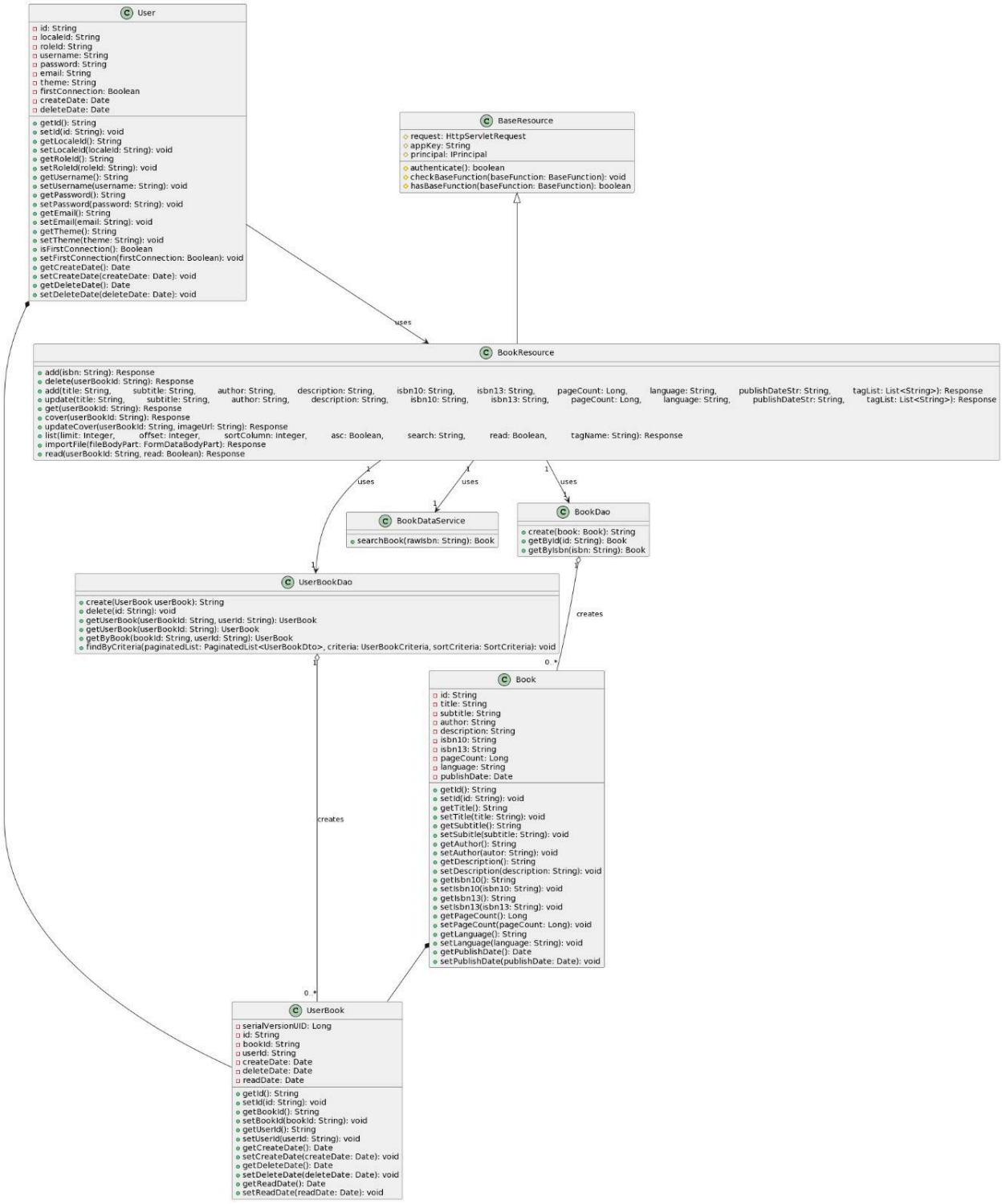
- Represents the mapping between a user and a book.
- Contains information such as the IDs of the user and the book, creation date, deletion date, etc.
- Provides methods to get and set these properties.

Book Class

- Represents a book entity in the system.
- Contains properties like title, author, description, ISBN, etc.
- Provides methods to get and set these properties.

BookDataService Class

- Handles the integration with external services (GoogleLibrary, OpenLibrary) to retrieve book information.
- Responsible for searching and fetching book data from external sources based on ISBN.
- Provides a method to search for book data using an ISBN.


User
- id: String
- localeId: String
- roleId: String
- username: String
- password: String
- email: String
- theme: String
- firstConnection: Boolean
- createDate: Date
- deleteDate: Date
+ getId(): String
+ setId(id: String): void
+ getLocaleId(): String
+ setLocaleId(localeId: String): void
+ getRoleId(): String
+ setRoleId(roleId: String): void
+ getUsername(): String
+ setUsername(username: String): void
+ getPassword(): String
+ setPassword(password: String): void
+ getEmail(): String
+ setEmail(email: String): void
+ getTheme(): String
+ setTheme(theme: String): void
+ isFirstConnection(): Boolean
+ setFirstConnection(firstConnection: Boolean): void
+ getCreateDate(): Date
+ setCreateDate(createDate: Date): void
+ getDeleteDate(): Date
+ setDeleteDate(deleteDate: Date): void
BaseResource
request: HttpServletRequest
appKey: String
principal: IPrincipal
authenticate(): boolean
checkBaseFunction(baseFunction: BaseFunction): void
hasBaseFunction(baseFunction: BaseFunction): boolean
uses
BookResource
+ add(isbn: String): Response
+ delete(userBookId: String): Response
+ add(title: String, subtitle: String, author: String, description: String, isbn10: String, isbn13: String, pageCount: Long, language: String, publishDateStr: String, tagList: List<String>): Response
+ update(title: String, subtitle: String, author: String, description: String, isbn10: String, isbn13: String, pageCount: Long, language: String, publishDateStr: String, tagList: List<String>): Response
+ get(userBookId: String): Response
+ cover(userBookId: String): Response
+ updateCover(userBookId: String, imageUrl: String): Response
+ list(limit: Integer, offset: Integer, sortColumn: Integer, asc: Boolean, search: String, read: Boolean, tagName: String): Response
+ importFile(fileBodyPart: FormDataBodyPart): Response
+ read(userBookId: String, read: Boolean): Response
1
uses
1
uses
1
uses
BookDataService
+ searchBook(rawIsbn: String): Book
BookDao
+ create(book: Book): String
+ getById(id: String): Book
+ getByIsbn(isbn: String): Book
UserBookDao
+ create(UserBook userBook): String
+ delete(id: String): void
+ getUserBook(userBookId: String, userId: String): UserBook
+ getUserBook(userBookId: String): UserBook
+ getByBook(bookId: String, userId: String): UserBook
+ findByCriteria(paginatedList: PaginatedList<UserBookDto>, criteria: UserBookCriteria, sortCriteria: SortCriteria): void
creates
1
0..*
creates
Book
- id: String
- title: String
- subtitle: String
- author: String
- description: String
- isbn10: String
- isbn13: String
- pageCount: Long
- language: String
- publishDate: Date
+ getId(): String
+ setId(id: String): void
+ getTitle(): String
+ setTitle(title: String): void
+ getSubtitle(): String
+ setSubitle(subtitle: String): void
+ getAuthor(): String
+ setAuthor(autor: String): void
+ getDescription(): String
+ setDescription(description: String): void
+ getIsbn10(): String
+ setIsbn10(isbn10: String): void
+ getIsbn13(): String
+ setIsbn13(isbn13: String): void
+ getPageCount(): Long
+ setPageCount(pageCount: Long): void
+ getLanguage(): String
+ setLanguage(language: String): void
+ getPublishDate(): Date
+ setPublishDate(publishDate: Date): void
1
0..*
UserBook
- serialVersionUID: Long
- id: String
- bookId: String
- userId: String
- createDate: Date
- deleteDate: Date
- readDate: Date
+ getId(): String
+ setId(id: String): void
+ getBookId(): String
+ setBookId(bookId: String): void
+ getUserId(): String
+ setUserId(userId: String): void
+ getCreateDate(): Date
+ setCreateDate(createDate: Date): void
+ getDeleteDate(): Date
+ setDeleteDate(deleteDate: Date): void
+ getReadDate(): Date
+ setReadDate(readDate: Date): void

UML Description 1B: Bookshelf Management Subsystem

User Class

- Represents a registered user in the system.
- Manages user properties such as ID, username, password, email, etc.
- Provides methods to get and set these properties.

BookResource Class

- Facilitates user interaction with book-related functionalities.
- Allows users to add, delete, update, retrieve, and manage books.
- Acts as the primary interface for users regarding book operations.

UserBookDao Class

- Handles storage and retrieval of user-book mappings.
- Stores the relationship between users and the books they own.
- Provides methods for creating, retrieving, and deleting user-book mappings.

UserBook Class

- Represents a particular user-book mapping.
- Contains information such as the IDs of the user and the book, creation date, deletion date, etc.
- Provides methods to get and set these properties.

TagResource Class

- Manages tags (bookshelves) created by users.
- Allows users to list, add, update, and delete tags.
- Acts as the interface for tag-related operations.

Tag Class

- Represents a tag (bookshelf) entity.
- Contains properties like ID, name, color, etc.

- Provides methods to get and set these properties.

TagDao Class

- Manages tag objects and their mappings.
- Handles CRUD operations for tags and their associations with user-books.
- Provides methods for retrieving, creating, updating, and deleting tags.

TagDto Class

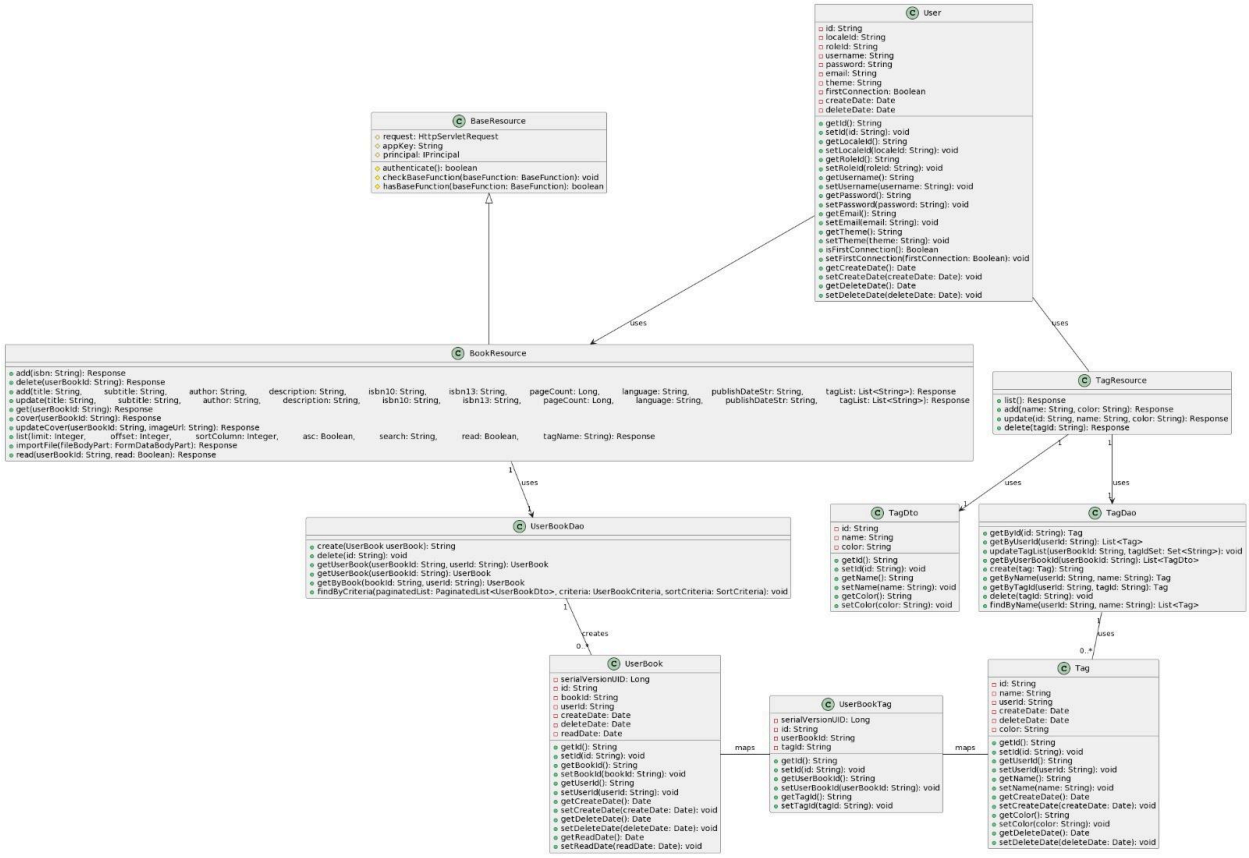
- Defines the structure of a tag when transferring it between modules of the system.
- Contains properties like ID, name, and color.

UserBookTag Class

- Maintains the mapping between user-books and tags associated with them.
- Represents a particular mapping between a user-book and a tag.
- Provides methods to get and set the mapping properties.

Relationships

- User uses BookResource to interact with book-related functionalities.
- BookResource uses UserBookDao to manage user-book mappings.
- UserBookDao creates and maintains UserBook instances.
- User uses TagResource to manage tags.
- TagResource uses TagDao to perform CRUD operations on tags.
- TagDao uses Tag to create and manage tag instances.
- TagResource also uses TagDto to transfer tag information between modules.
- UserBook and Tag are mapped by UserBookTag to signify the association between user-books and tags.



UML Description 1C: User Management Sub-system

User

- Represents a user entity in the system.
- Stores user information such as id, locale, role, username, password, email, theme, etc.
- Provides methods to access and modify user attributes.

BaseResource

- Provides base functionality and utilities for resource classes.
- Contains methods for handling authentication, authorization, etc.
- Authenticates the user making the request.
- Checks if a base function is available for the user.
- Checks if the user has access to a specific base function.

UserResource

- Handles HTTP requests related to user management.
- Provides endpoints for user registration, login, logout, updating user information, etc.
- Interacts with UserDao and AuthenticationTokenDao for database operations.
- Methods for handling user registration, login, logout, updating user information, etc.
- Validates input parameters, interacts with database layers for data operations**.**

UserDao

- Performs CRUD (Create, Read, Update, Delete) operations on user data in the database.
- Provides methods for user authentication, creation, updating, deletion, and retrieval.
- Handles operations related to user data management.
- Authenticates a user based on username and password.
- Creates a new user in the database.
- Updates an existing user's information.
- Deletes a user from the system.
- Retrieves a list of users with pagination and sorting.

UserDto

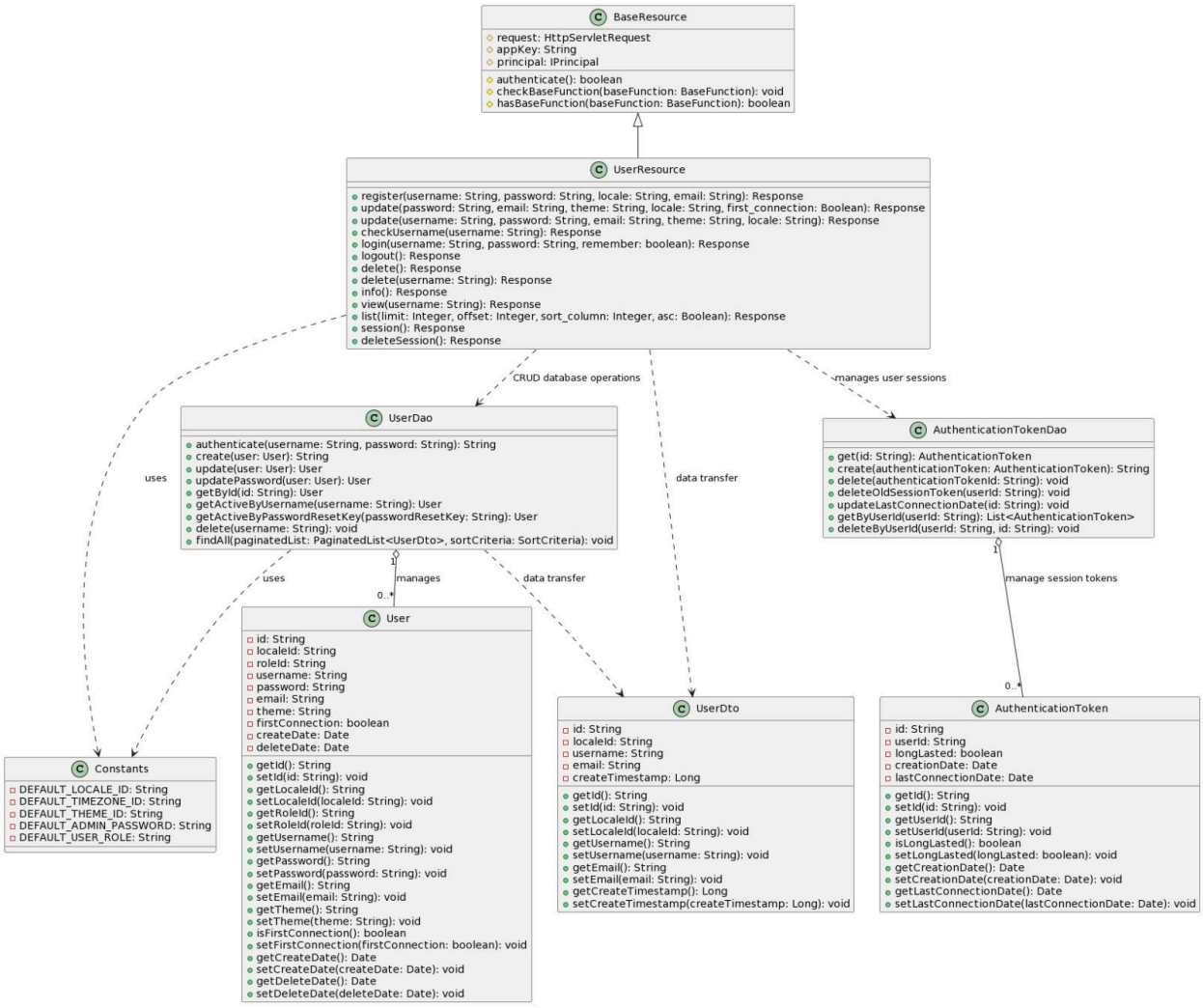
- Represents a Data Transfer Object (DTO) for user-related data.
- Used for transferring user data between different layers of the application.
- Getter and setter methods for various user-related attributes.
- Transfers user data between the service and resource layers.

AuthenticationTokenDao

- Performs CRUD operations on authentication tokens in the database.
- Manages authentication tokens associated with user sessions.
- Provides methods for token creation, deletion, retrieval, etc.
- Retrieves an authentication token by its ID.
- Creates a new authentication token.
- Deletes an authentication token from the system.
- Provides methods for managing token expiration, updating last connection date, etc.

AuthenticationToken

- Represents an authentication token associated with a user session.
- Stores information about the token ID, user ID, creation date, last connection date, etc.
- Used for managing user sessions and authentication.
- Retrieves the authentication token's ID.
- Sets the authentication token's ID.
- Retrieves the user ID associated with the token.
- Sets the user ID associated with the token.
- Methods to handle creation date, last connection date, and other attributes.



Observations:

Strengths:

- **Modular Design**: Each class represents a specific module or functionality, promoting modularity and ease of maintenance. For example, UserResource handles HTTP requests related to user management, while UserDao manages CRUD operations on user data in the database.
- **Clear Separation of Responsibilities**: The system demonstrates a clear separation of concerns with distinct classes responsible for

specific functionalities like user management, authentication, database operations, and resource handling.

- **Database Abstraction**: The DAO classes abstract database operations, encapsulating the details of interaction with the database. This promotes scalability and maintainability by allowing changes to the underlying database without affecting higher-level modules.
- **Data Transfer Objects (DTOs)**: The use of UserDto as a Data Transfer Object helps in transferring user-related data between different layers of the application, enhancing encapsulation and reducing coupling between layers.

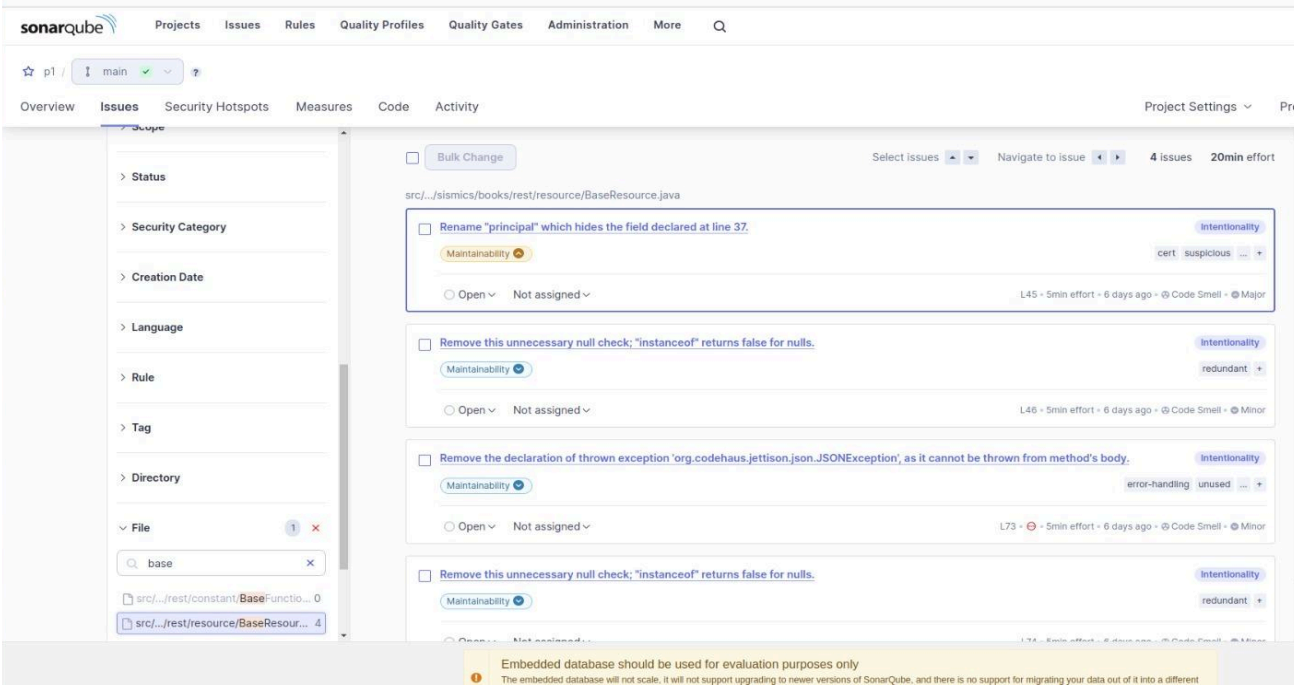
Weaknesses:

- **Complex Methods**: BookResource class seems to have high complexity due to many methods and various responsibilities. There are multiple methods with similar names but different parameters, such as update and delete, which introduces complexity and ambiguity.
- **Violation of SRP**: BookResource class appears to handle not only routing HTTP requests but also validation and potentially business logic related to user management. This can lead to bloated resource classes that are responsible for too many tasks, violating the Single Responsibility Principle (SRP).

Task 2A: SonarQube for Smell Detection

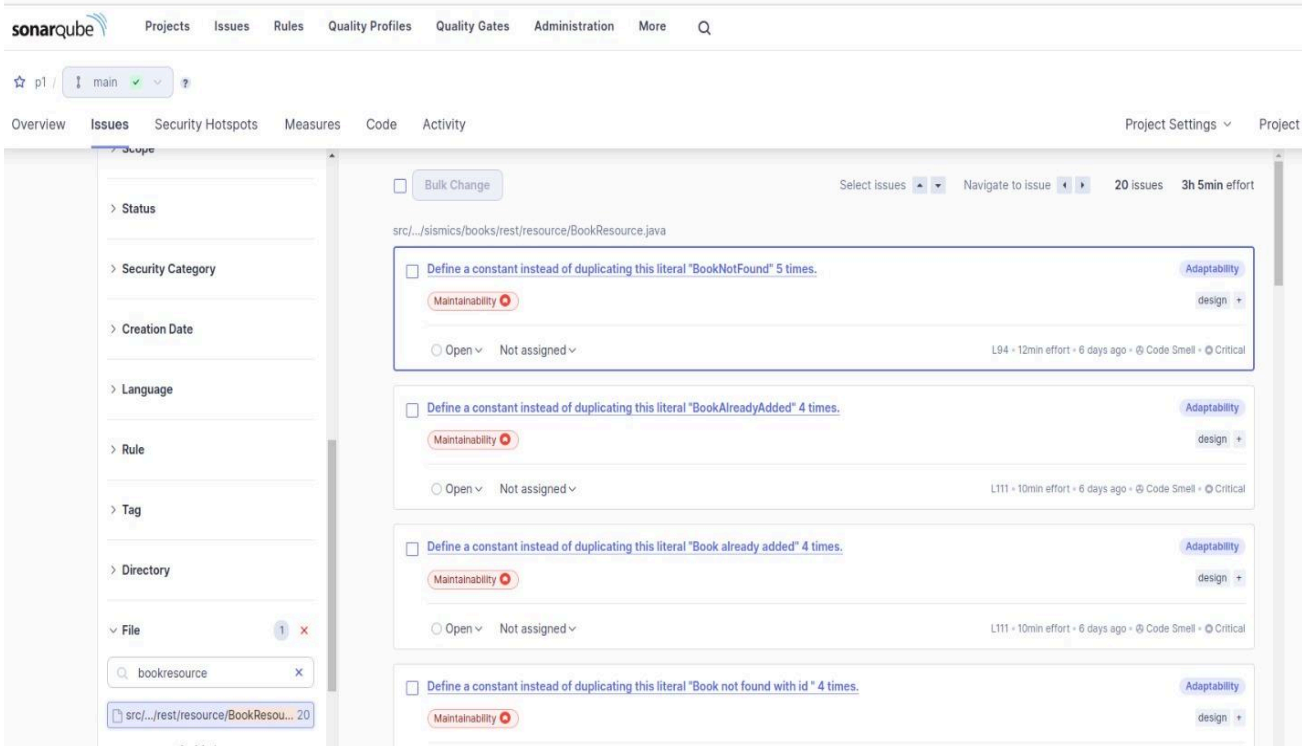
BaseResource

1. Unutilized abstraction:
   BaseResource is declared as an abstract class but contains no abstract functions to be implemented.
   Below are the code smells detected by Sonarqube in BaseResource class.



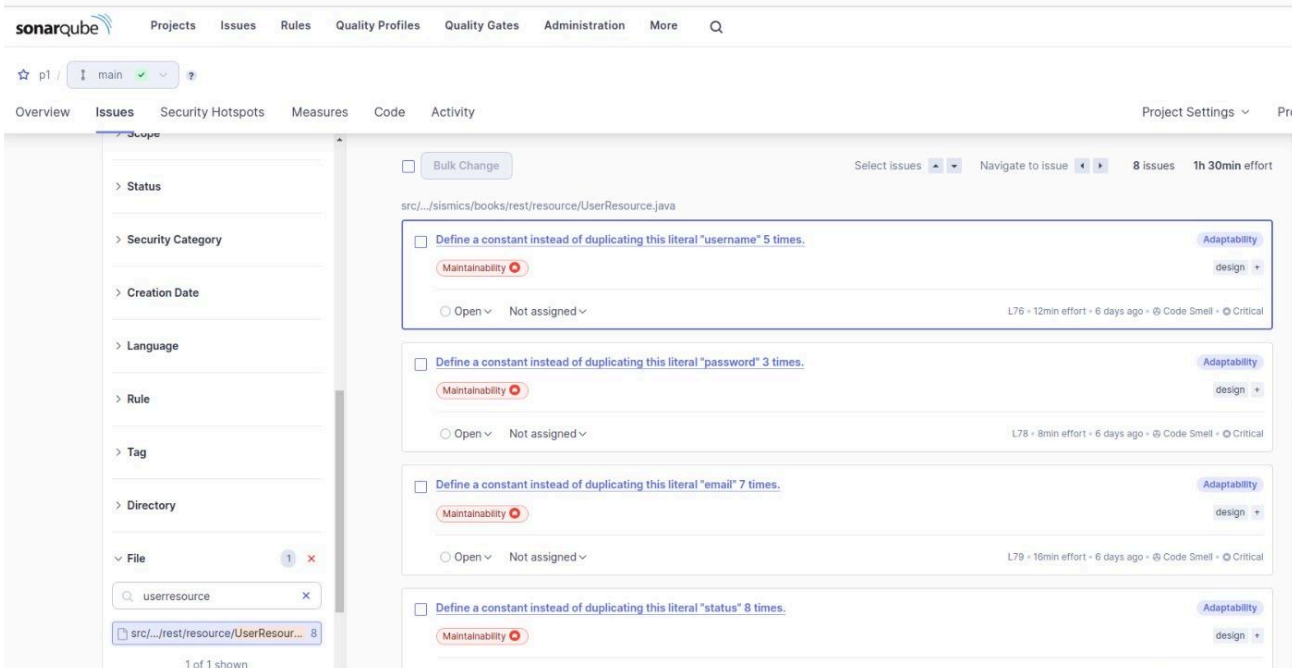
BookResource

1. God class
   BookResource acts as API endpoint for the Books but is handling many things which defeats the purpose of single responsibility.
2. Broken hierarchy
   BookResource unnecessarily extends BaseResource where authentication is being done which is not the responsibility of the BookResource class.
   Below are the code smells detected by Sonarqube in BookResource class.



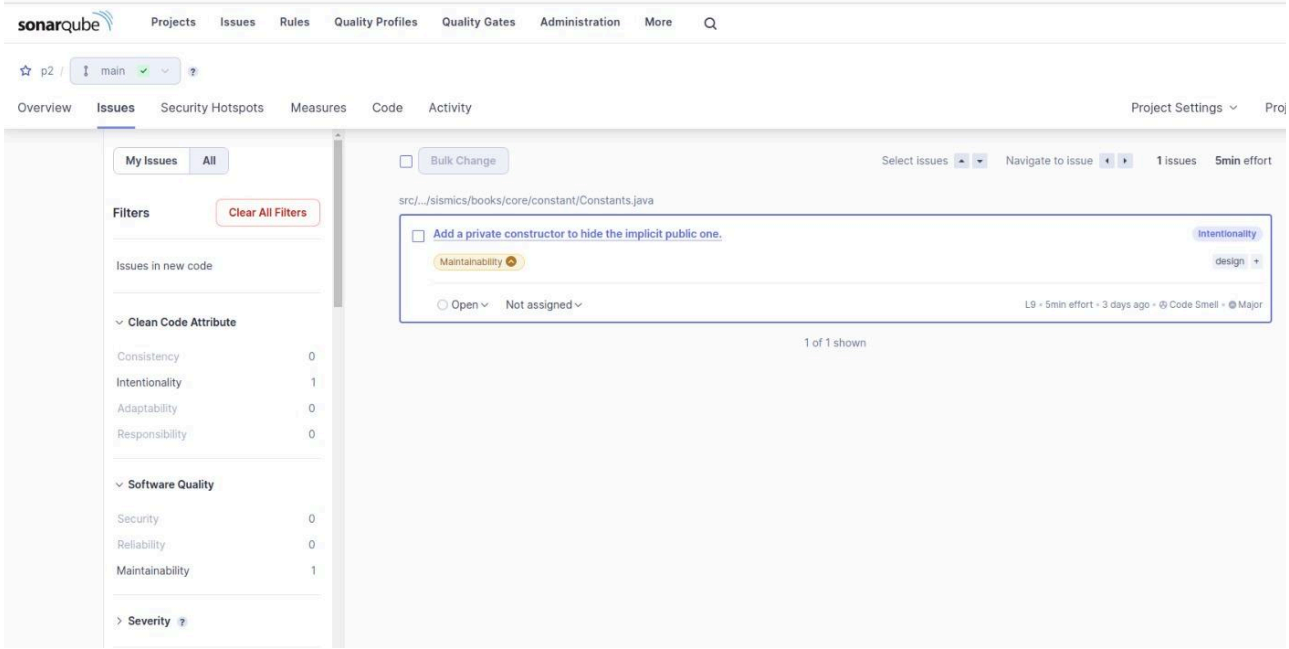
UserResource

1. God class
   UserResource acts as API endpoint for the User management but is handling many things which defeats the purpose of single responsibility.
2. Broken hierarchy
   UserResource unnecessarily extends BaseResource where authentication is being done which is not the responsibility of UserResource class.
   Below are the code smells detected by Sonarqube in UserResource class.

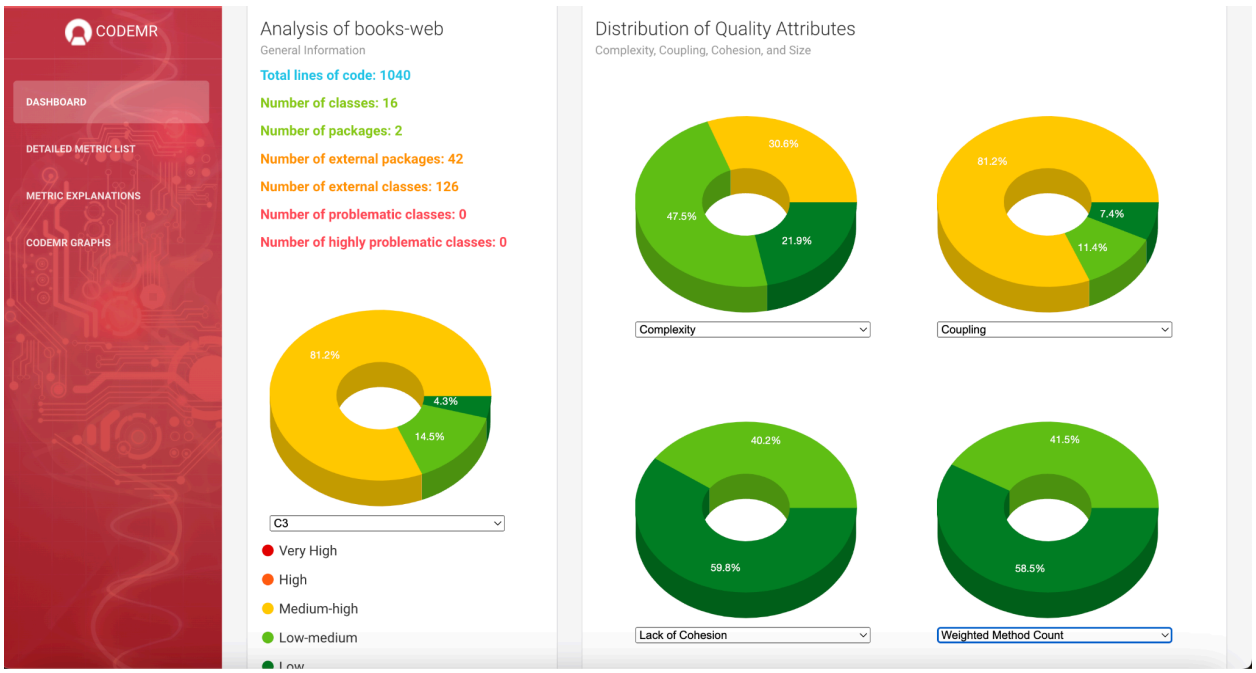


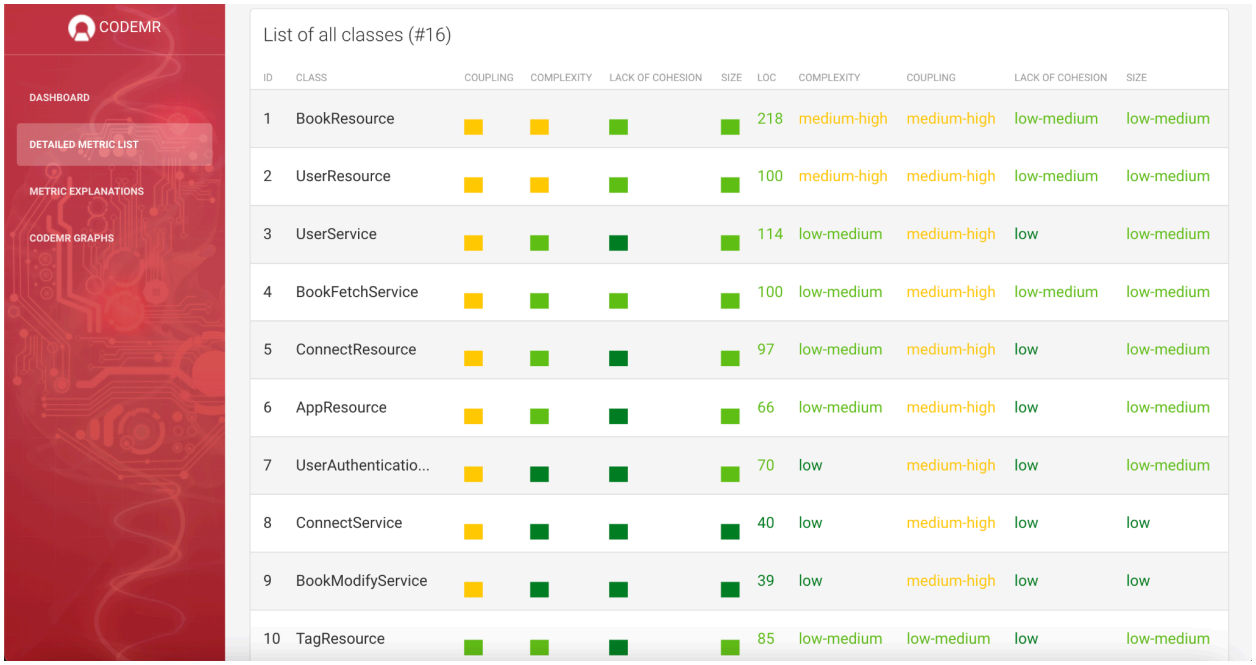
Constants

1. Deficient encapsulation: All the constants are placed in a single Constants file which exposes its internal state (constant values) directly to the outside world through public fields. This violates the principle of encapsulation, which suggests hiding internal state and providing controlled access to it through methods.
   Below are the code smells detected by Sonarqube in Constant class.

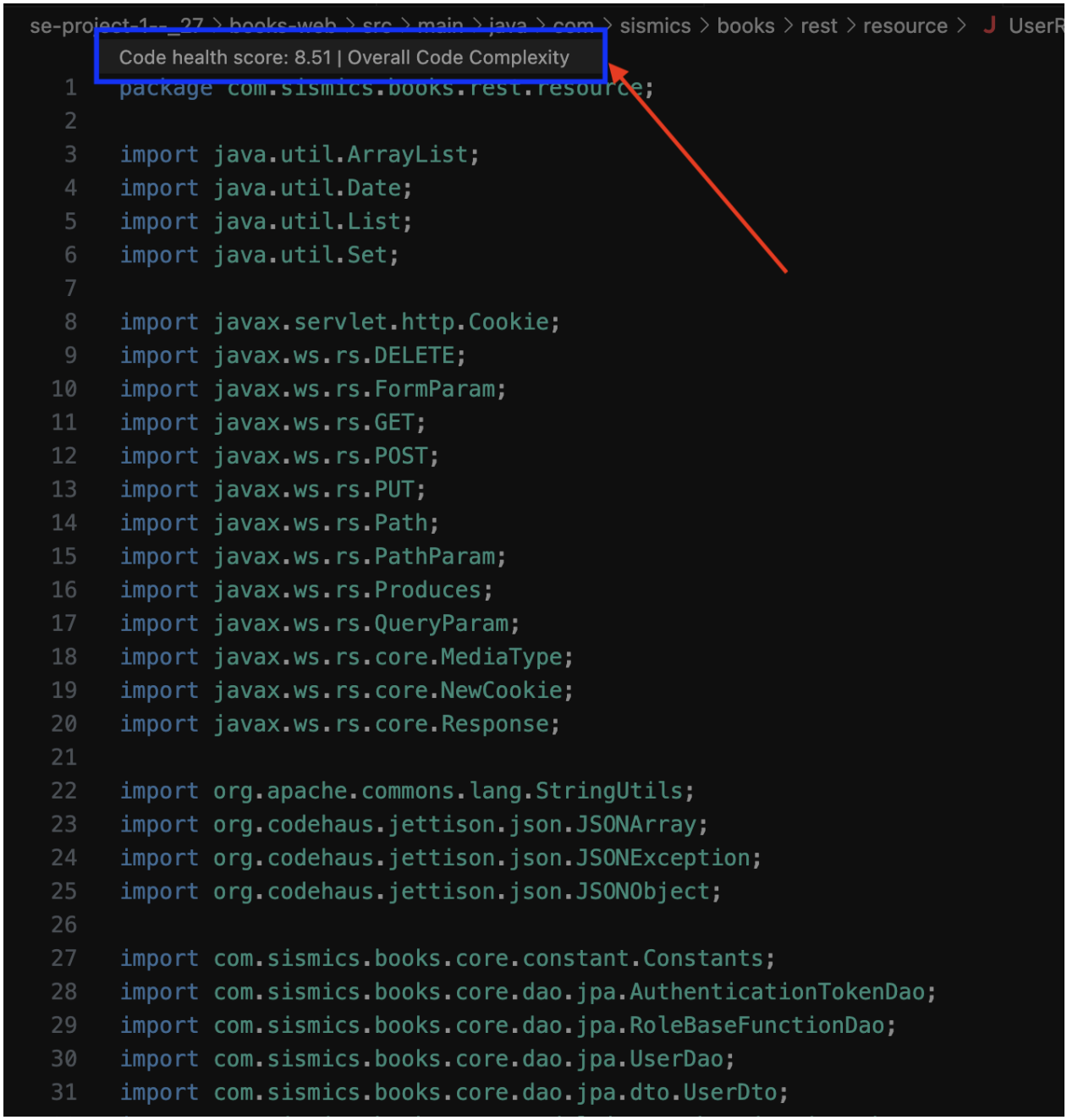


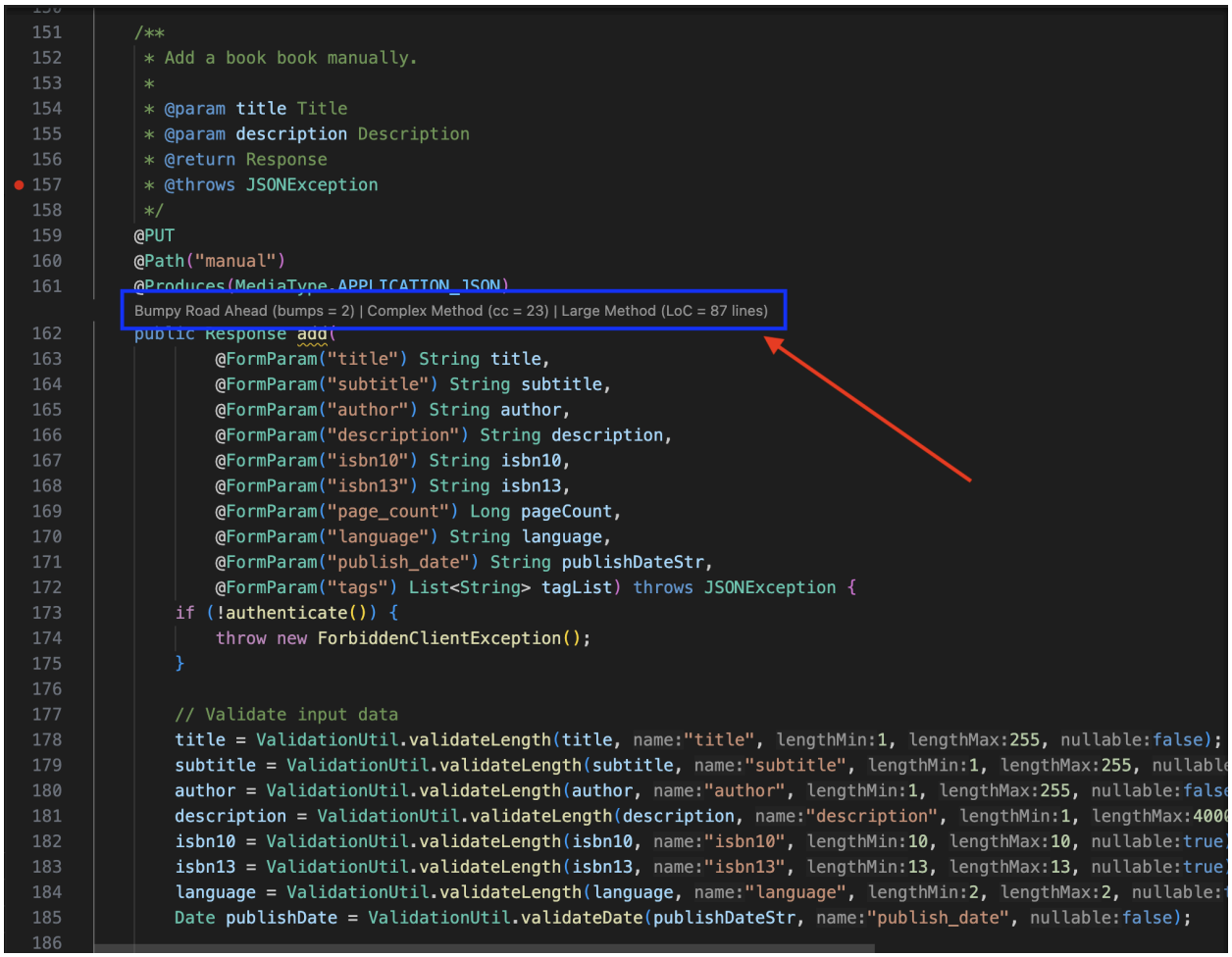
Task 2B: Comprehensive Code Metrics Analysis



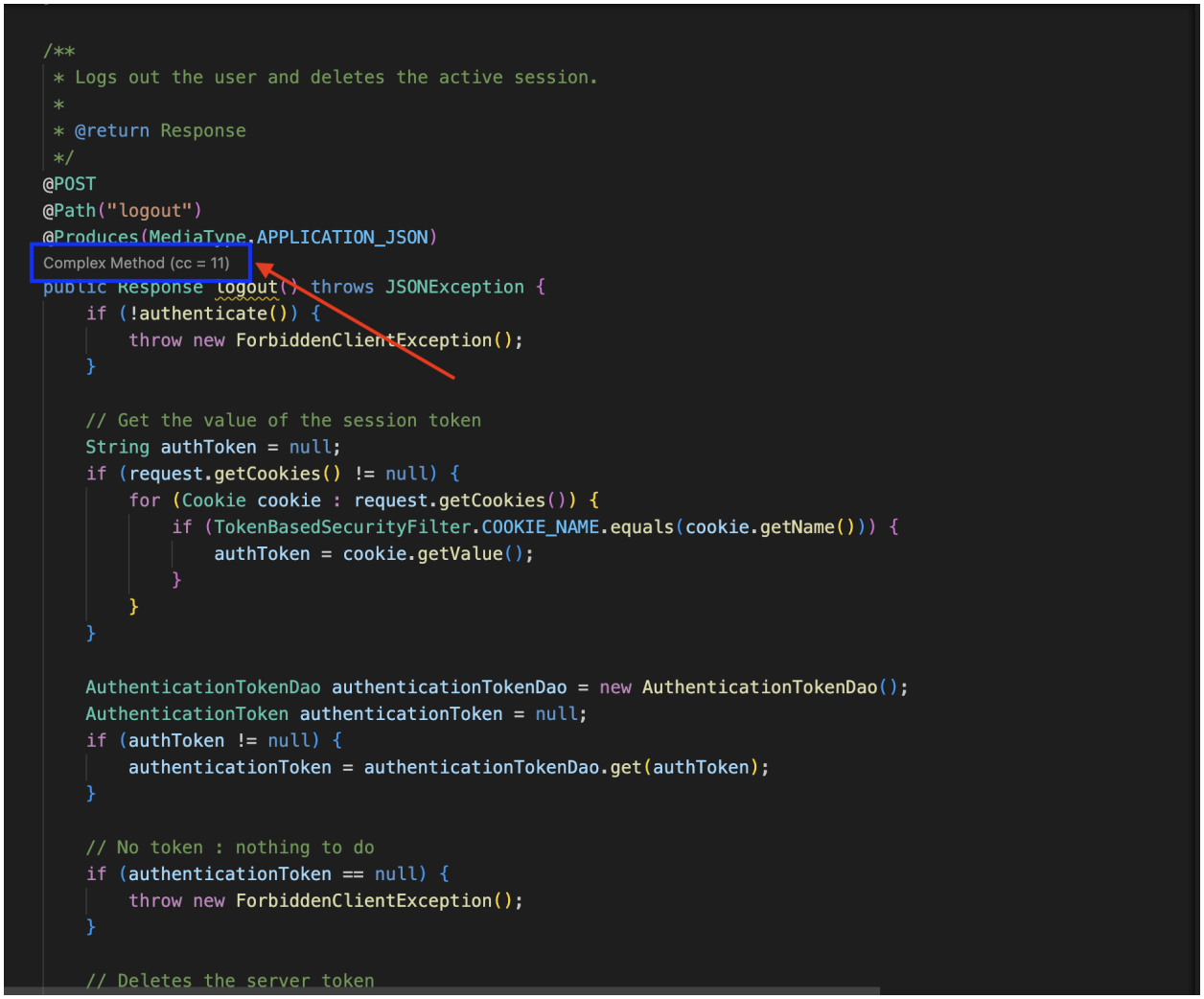


ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	BookResource					218	medium-high	medium-high	low-medium	low-medium
2	UserResource					100	medium-high	medium-high	low-medium	low-medium
3	UserService					114	low-medium	medium-high	low	low-medium
4	BookFetchService					100	low-medium	medium-high	low-medium	low-medium
5	ConnectResource					97	low-medium	medium-high	low	low-medium
6	AppResource					66	low-medium	medium-high	low	low-medium
7	UserAuthenticatio...					70	low	medium-high	low	low-medium
8	ConnectService					40	low	medium-high	low	low
9	BookModifyService					39	low	medium-high	low	low
10	TagResource					85	low-medium	low-medium	low	low-medium


se-project-1--_27 > books-web > src > main > java > com > sismics > books > rest > resource > J UserR
Code health score: 8.51 | Overall Code Complexity
1 package com.sismics.books.rest.resource;
2
3 import java.util.ArrayList;
4 import java.util.Date;
5 import java.util.List;
6 import java.util.Set;
7
8 import javax.servlet.http.Cookie;
9 import javax.ws.rs.DELETE;
10 import javax.ws.rs.FormParam;
11 import javax.ws.rs.GET;
12 import javax.ws.rs.POST;
13 import javax.ws.rs.PUT;
14 import javax.ws.rs.Path;
15 import javax.ws.rs.PathParam;
16 import javax.ws.rs.Produces;
17 import javax.ws.rs.QueryParam;
18 import javax.ws.rs.core.MediaType;
19 import javax.ws.rs.core.NewCookie;
20 import javax.ws.rs.core.Response;
21
22 import org.apache.commons.lang.StringUtils;
23 import org.codehaus.jettison.json.JSONArray;
24 import org.codehaus.jettison.json.JSONException;
25 import org.codehaus.jettison.json.JSONObject;
26
27 import com.sismics.books.core.constant.Constants;
28 import com.sismics.books.core.dao.jpa.AuthenticationTokenDao;
29 import com.sismics.books.core.dao.jpa.RoleBaseFunctionDao;
30 import com.sismics.books.core.dao.jpa.UserDao;
31 import com.sismics.books.core.dao.jpa.dto.UserDto;




```
151     /**
152      * Add a book book manually.
153      *
154      * @param title Title
155      * @param description Description
156      * @return Response
157      * @throws JSONException
158      */
159     @PUT
160     @Path("manual")
161     @Produces(MediaType.APPLICATION_JSON)
162     public Response add(
163             @FormParam("title") String title,
164             @FormParam("subtitle") String subtitle,
165             @FormParam("author") String author,
166             @FormParam("description") String description,
167             @FormParam("isbn10") String isbn10,
168             @FormParam("isbn13") String isbn13,
169             @FormParam("page_count") Long pageCount,
170             @FormParam("language") String language,
171             @FormParam("publish_date") String publishDateStr,
172             @FormParam("tags") List<String> tagList) throws JSONException {
173         if (!authenticate()) {
174             throw new ForbiddenClientException();
175         }
176
177         // Validate input data
178         title = ValidationUtil.validateLength(title, name:"title", lengthMin:1, lengthMax:255, nullable:false);
179         subtitle = ValidationUtil.validateLength(subtitle, name:"subtitle", lengthMin:1, lengthMax:255, nullable
180         author = ValidationUtil.validateLength(author, name:"author", lengthMin:1, lengthMax:255, nullable:false
181         description = ValidationUtil.validateLength(description, name:"description", lengthMin:1, lengthMax:4000
182         isbn10 = ValidationUtil.validateLength(isbn10, name:"isbn10", lengthMin:10, lengthMax:10, nullable:true)
183         isbn13 = ValidationUtil.validateLength(isbn13, name:"isbn13", lengthMin:13, lengthMax:13, nullable:true)
184         language = ValidationUtil.validateLength(language, name:"language", lengthMin:2, lengthMax:2, nullable:f
185         Date publishDate = ValidationUtil.validateDate(publishDateStr, name:"publish_date", nullable:false);
186
```



Tools Used for Metric Analysis are as follows:

1) CodeMR - It is a static code analysis tool that helps developers visualize and understand the complexities of their software architecture. It provides metrics on coupling, cohesion, and other quality attributes. Two key points about CodeMR are:

- It offers a comprehensive visual representation of the codebase, making it easier to identify and address architectural and design issues.
- The metrics provided by CodeMR can be a strong indicator of potential maintainability issues, allowing teams to prioritize refactoring efforts.

2) CodeScene - It is a tool that provides insights into the evolution of a codebase, focusing on the organizational side of code. Two significant aspects of CodeScene are:

- It analyzes the 'hotspots' of a codebase, identifying areas that might be prone to errors or require more frequent changes, guiding the refactoring process.
- CodeScene integrates social data from the codebase, such as code ownership and team interaction, which can be crucial for understanding the impact of team dynamics on the code quality.

Implications Discussion:

- **Complexity** - High complexity can indicate that a class or method may be trying to do too much, making it harder to understand, test, and maintain. Reducing complexity often involves refactoring into simpler methods or classes.

- **Coupling** - High coupling suggests classes are interdependent, making changes more difficult and error-prone. Lower coupling is desirable for easier maintainability and flexibility.

- **Cohesion** - Low cohesion can imply that a class has multiple unrelated responsibilities and may need to be broken down into more focused, cohesive components.

- **Size (LOC - Lines of Code)** - Large classes or methods can be challenging to maintain. They may need to be broken down into smaller, more manageable pieces.

- **Number of Problematic Classes -** This indicates classes that may need immediate attention due to their violation of good coding practices, potentially impacting maintainability and performance.

- **Weighted Method Count** - The Weighted Method Count metric is defined as the sum of complexities of all methods declared in a class. This metric is a good indicator how much effort will be necessary to maintain and develop a particular class.

Task 3C: Leveraging Large Language Models for Refactoring

1 - Identify Code Snippets: The identified code snippet which describes the code smells is attached in part 2A.

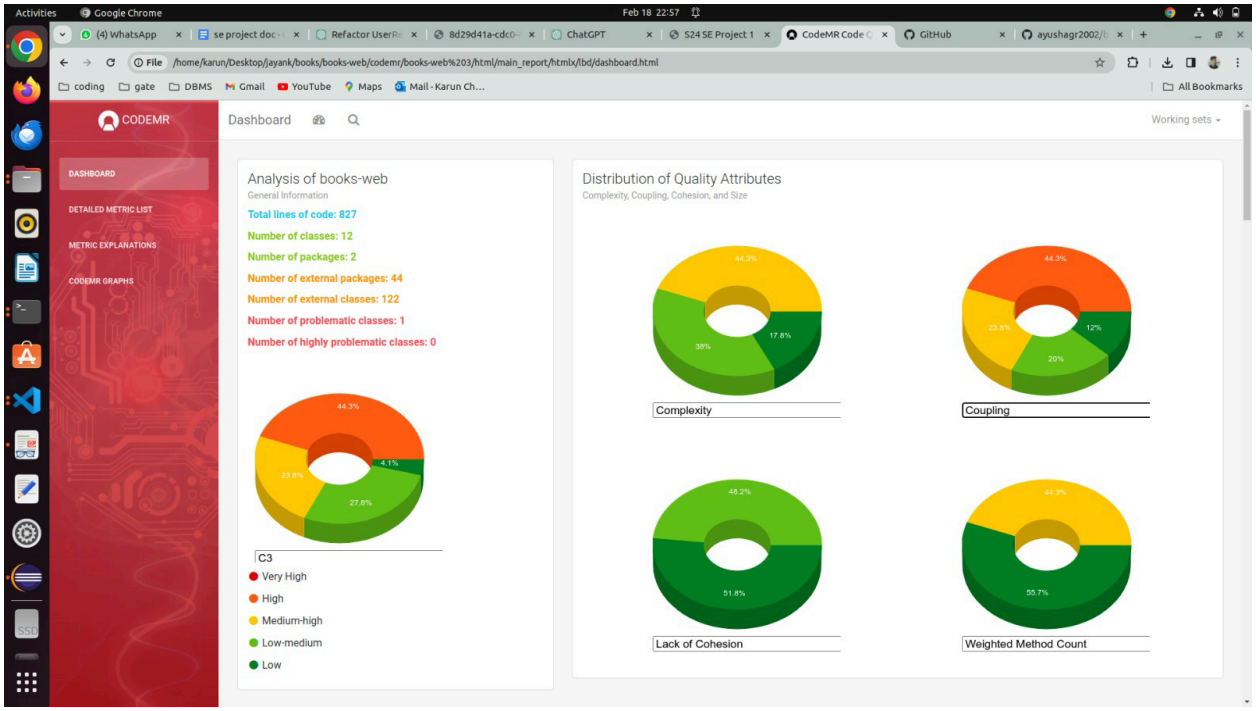
2 - Apply Manual Refactoring: The appropriate refactored code is pushed to the github by identifying the smells and dividing them into different issues and later pushing the refactored code solution for those issues.

3 - LLM Refactoring: Below, we have provided the link which consists of all the conversation regarding finding design smells and the appropriate refactoring code given by the LLMs.

Here are the links: link1 and link2.

4 - Evaluate and Document:
Document the differences between the manual and LLM-generated refactored versions.

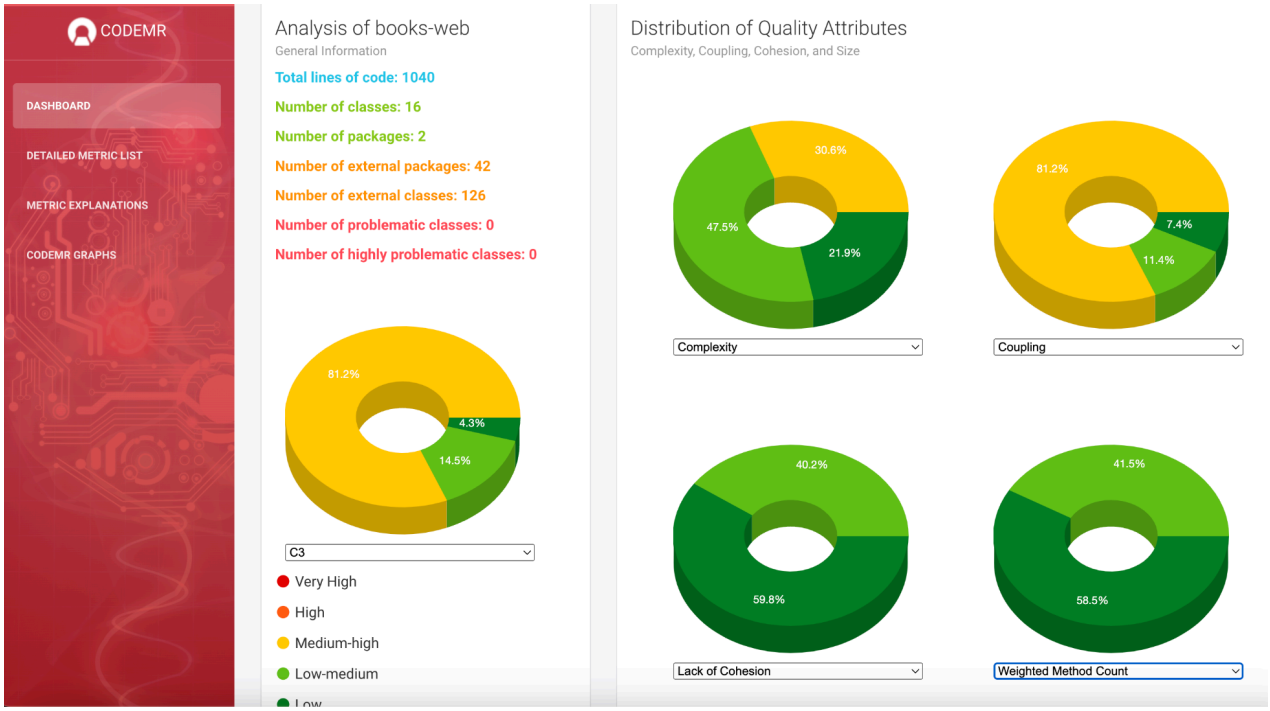
LLM Generated Refactored Code Metric Analysis:

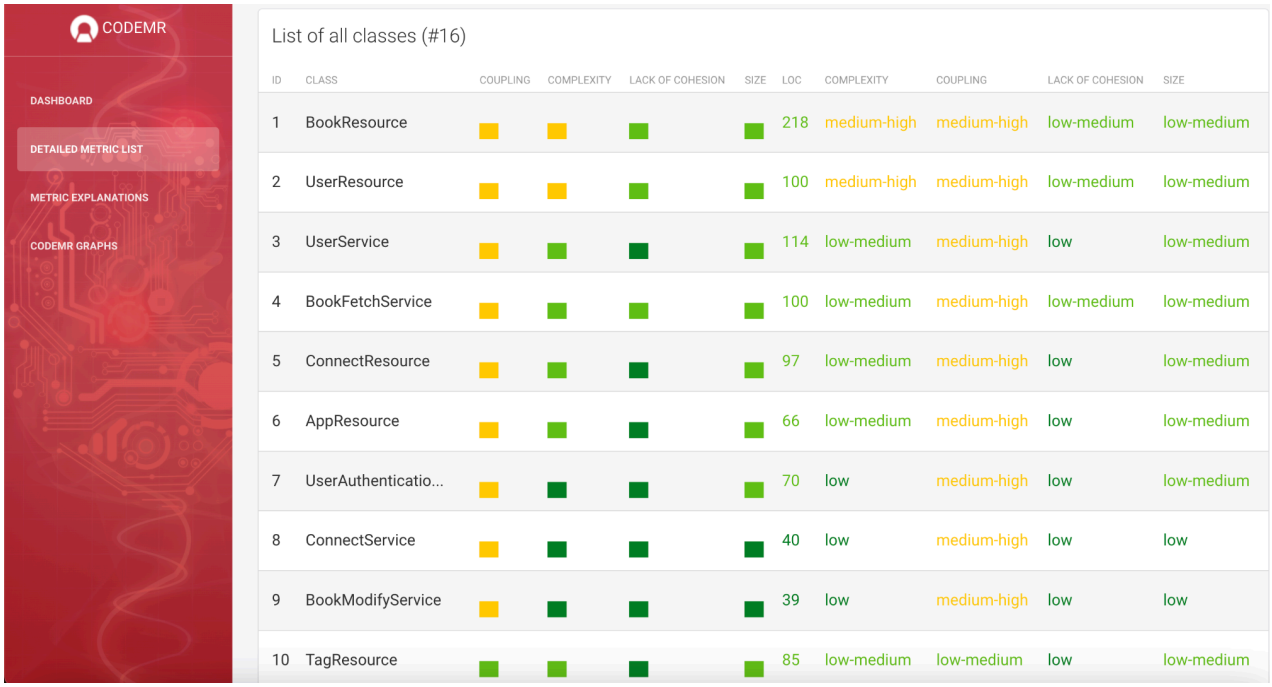


List of all classes (#12)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	BookResource					366	medium-high	high	low-medium	medium-high
2	ConnectResource					131	low-medium	medium-high	low	low-medium
3	AppResource					66	low-medium	medium-high	low	low-medium
4	TagResource					85	low-medium	low-medium	low	low-medium
5	UserResource					80	low	low-medium	low	low-medium
6	ThemeResource					15	low-medium	low	low	low
7	LocaleResource					15	low-medium	low	low	low
8	BaseFunction					2	low-medium	low	low	low
9	BaseResource					33	low	low	low-medium	low
10	TextPlainMessageB...					24	low	low	low	low
11	UserUpdateRequest					6	low	low	low	low
12	UserRegistrationR...					4	low	low	low	low

Manually Refactored Code Metric Analysis:





ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	BookResource					218	medium-high	medium-high	low-medium	low-medium
2	UserResource					100	medium-high	medium-high	low-medium	low-medium
3	UserService					114	low-medium	medium-high	low	low-medium
4	BookFetchService					100	low-medium	medium-high	low-medium	low-medium
5	ConnectResource					97	low-medium	medium-high	low	low-medium
6	AppResource					66	low-medium	medium-high	low	low-medium
7	UserAuthenticatio...					70	low	medium-high	low	low-medium
8	ConnectService					40	low	medium-high	low	low
9	BookModifyService					39	low	medium-high	low	low
10	TagResource					85	low-medium	low-medium	low	low-medium

Manual Refactoring Analysis:

- Total Lines of Code: There are 1040 lines of code in the manually refactored version.
- Classes and Packages: The project consists of 16 classes and 2 packages, with a significant number of external packages and classes used.
- Code Quality Attributes: The distribution of quality attributes such as complexity, coupling, cohesion, and size are visualized in donut charts. Most of the code seems to maintain good levels of cohesion and complexity, while coupling and method count show areas with higher values, which could indicate potential areas for improvement.
- Classes Metrics: The classes have varying levels of complexity and coupling, with some classes showing medium-high complexity and medium-high coupling, which could be a concern for maintainability.

LLM-GPT Refactoring Analysis -

- Total Lines of Code: The LLM-GPT refactored version shows a reduction in the total lines of code to 827.
- Classes and Packages: The number of classes has decreased to 12, with an increase in external packages and classes.
- Code Quality Attributes: The donut charts indicate that while there is an improvement in complexity, there seems to be a slight increase in coupling, which may not be desirable. Cohesion and method counts are slightly improved.
- Classes Metrics: The LLM-GPT refactored classes tend to have lower complexity and better cohesion, although the coupling in some classes has increased, which could lead to less maintainable code if not addressed.

Strengths and Weaknesses:

1. **Manual Refactoring**

- **Strengths:** Manual refactoring allows for nuanced changes that a developer can make based on deep understanding of the codebase and its context. It often leads to more thoughtful restructuring and potential architectural improvements.
- **Weaknesses:** It can be time-consuming and dependent on the refactoring skills of the developer. There is also the risk of human error, and it may lack consistency across different parts of the codebase.

2. **LLM-GPT Refactoring**

- **Strengths:** LLM-GPT refactoring can process large codebases quickly and provide refactoring suggestions at scale. It can identify patterns and apply refactoring uniformly across the codebase.
- **Weaknesses:** It may lack understanding of the business context and could introduce changes that improve certain metrics but worsen the overall architecture. It may also overfit to certain coding styles or patterns it was trained on, potentially leading to less creative solutions.

3. **Scenarios Where LLMs Excel**

- Bulk Refactoring: LLMs can quickly refactor large amounts of boilerplate code or apply consistent coding standards across a codebase.
- Pattern Recognition: LLMs are good at identifying common anti-patterns and suggesting standard refactoring techniques.

4. **Scenarios Where Manual Intervention is Preferable**

- Complex Architectural Changes: Deep architectural changes often require a developer's insight and understanding of the system's design and goals.

- Business Logic: Refactoring that involves complex business logic might need a developer's expertise to ensure that the logic remains correct and clear.

Conclusion:

Both manual and LLM-GPT refactoring have their place in software development. LLMs can provide valuable assistance, especially in large codebases where manual refactoring is impractical. However, for critical systems or when deep understanding is required, manual refactoring remains indispensable. A combined approach, where LLM suggestions are reviewed and potentially integrated by experienced developers, might offer the best balance between speed and quality.

Project Contributions:
Vivek Ram:
Created UML for Book Addition & Display Subsystem and Bookshelf management system.
BaseResource - Identified and refactored unnecessary abstract class.
BookResource - Identified and refactored code to reduce bulk of BookResource class and distribute functionality into logically separate methods(BookModifyService.java).

Jayank Mahaur:
Ran CodeMR for code metrics analysis prior to refactoring and after refactoring the code.
BookResource - Identified and refactored code to reduce bulk of BookResource class and distribute functionality into logically separate methods(BookFetchService.java).

Ronak Redkar:
Created UML Class diagram for User Management Subsystem
UserResource - Identified and refactored class to reduce complexity, long methods and handle violation of SRP.
Created a new class (UserService) to handle CRUD operations from UserResource to separate methods.
BookResource and Book - Updated BookResource Class and Book Class to reduce cyclomatic complexity.

Naitik Kariwal:
Ran CodeMR for code metrics analysis prior to refactoring and after refactoring the code.
Created a new Class - UserSessionService that handles session management
Updated UserResource Class to reduce the code complexity (long method) and violation of SRP

Karun Choudhary:
UserResource - Identified bulk of UserResource class and split it into logically separate class UserAuthenticationService to handle authentication related tasks.
Used Chatgpt to get LLM's refactored code and checked its metrics compared to manual refactoring metric.